ELSEVIER

Computational Geometry 4 (1994) 99–118

Computational Geometry
Theory and Applications

Extremal polygon containment problems*

Micha Sharir[a,b,*], Sivan Toledo[b]

[a]*Courant Institute of Mathematical Sciences, New York University, New York, NY 10012, USA*
[b]*School of Mathematical Sciences, Tel Aviv University, Ramat Aviv, 69978 Tel Aviv, Israel*

Communicated by Chee Yap; submitted 12 September 1991; accepted 25 August 1993

Abstract

Given a convex polygonal object P with k vertices and an environment consisting of polygonal obstacles with a total of n corners, we seek a placement for the largest copy of P that does not intersect any of the obstacles, allowing translation, rotation and scaling. We employ the parametric search technique of Megiddo (1983), and the fixed size polygon placement algorithms developed by Leven and Sharir (1987), to obtain an algorithm that runs in time $O(k^2 n\lambda_6(kn)\log^3(kn)\log\log(kn))$. We also present several other efficient algorithms for restricted variants of the extremal polygon containment problem, using the same ideas. These variants include: placement of the largest homothetic copies of one or two convex polygons in another convex polygon and placement of the largest similar copy of a triangle in a convex polygon.

Key words: Polygon containment; Motion planning; Parametric searching

1. Introduction

Let P be a convex polygon having k vertices and edges, and let Q be a closed two dimensional environment bounded by a collection of polygonal obstacles (the 'environment') having altogether n corners. The main problem solved in this paper is to compute the largest possible placement of a similar copy of P that can be placed inside

*A preliminary version of the results in this paper appeared in [26]. Work on this paper by the first author has been supported by Office of Naval Research Grant N00014-90-J-1284, by National Science Foundation Grant CCR-89-01484, and by grants from the U.S.-Israeli Binational Science Foundation, the G.I.F.—the German Israeli Foundation for Scientific Research and Development, and the Fund for Basic Research administered by the Israeli Academy of Sciences.

*Corresponding author.

SSDI 0925-7721(93)E0020-H

Q, that is, a placement in which the (translated, rotated and scaled) copy of P does not intersect any of the obstacles. We also give efficient algorithms that solve similar extremal polygon containment problems under more restrictive conditions, and an algorithm that computes largest disjoint placements of two polygons in a third.

Some papers study the *fixed-size* polygon containment problem, in which (the convex) P is only allowed to translate and rotate and we wish to determine whether there is any placement of a copy of P inside Q [6, 4].

Chazelle [6] studies the problem for the case where P and Q are arbitrary simple polygons and presents a naive algorithm that takes time $O(k^3n^3(k+n)\log(k+n))$. A more restricted case of the problem, in which both P and Q are convex, is also studied by Chazelle [6], who solves this case in time $O(kn^2)$. Chazelle gives a simple solution to an even more restricted version in which P is a triangle; this version runs in time $O(n^2)$. Avnaim and Boissonnat [5] present an algorithm for the case where both P and Q are non-convex, possibly non-connected polygons, which runs in time $O(k^3n^3\log(kn))$. In another paper Avnaim and Boissonnat [4] investigate the problem of simultaneous placement of two or three not necessarily convex polygons in a closed polygonal environment. For this problem they allow translations only.

Extremal polygon containment problems were also previously studied. Fortune [14], and Leven and Sharir [20] consider the following problem: find the largest homothetic copy of P inside Q. In other words, translation and scaling of P are allowed, but rotation is not. When P is convex and Q is an arbitrary polygonal environment, this problem is solved in time $O(kn\log(kn))$ by constructing a generalized Voronoi diagram of Q under a convex distance function induced by P.

Chew and Kedem [9] follow a related approach to solve a more difficult variant of the problem, in which P is also allowed to rotate, which is also the main problem studied in this paper. Instead of a Voronoi diagram, they compute the Delaunay triangulation of Q under the convex distance function induced by P at some arbitrary fixed orientation. By using a clever incremental technique for constructing all the topologically different triangulations obtained as the orientation of P varies, they solve the problem in time $O(k^4n\lambda_3(kn)\log n)$, where $\lambda_q(r)$ is the maximum length of an (r, q)-Davenport-Schinzel sequence (which is almost linear in r for any fixed q) [1, 16].

In this paper we follow a different approach that applies the parametric search technique introduced by Megiddo [21]. By exploiting efficient sequential and parallel algorithms for the fixed size containment problem, we solve the extremal problem in time $O(k^2n\lambda_6(kn)\log^3(kn)\log\log(kn))$.

There are two advantages of our technique over the technique of [9]. First, our solution is considerably faster than theirs when k is large—roughly two orders of magnitude faster. Second, the application of Megiddo's technique to largest placement problems is so natural that it is surprising that no one has observed this connection before. Roughly speaking, a solution for the fixed-size problem allows us to determine whether any specified expansion ratio is too large or too small. This, plus an efficient parallel version of the fixed size containment algorithm, is all that is required for Megiddo's technique to apply (see below for more details). We demonstrate the

generality of our approach by considering several other extremal containment problems, and show that Megiddo's technique applies to all of them. Specifically we consider the extremal versions of the following problems: placing a convex polygon in another convex polygon under translation, placing two convex polygons in a third convex polygon under translation, placing a triangle in a convex polygon under translation and rotation, and finally the general case of placing a convex polygon in a polygonal environment under translation and rotation. In all cases, we obtain efficient algorithms whose running time differs from that of the corresponding algorithm for the fixed-size problem by only a poly-logarithmic factor. No algorithms with comparable running time were known previously. Some additional possible extensions of the technique are discussed at the end of the paper.

The paper is organized as follows. In Section 2 we investigate some simple versions of the extremal polygon containment problem, involving one and two polygons, and allowing the polygons only to translate. Although these problems can be solved efficiently using parametric searching, they are sufficiently simple to admit more direct and somewhat improved solutions. These improved methods are described in Section 3. Section 4 is devoted to a simple version of the general case, in which we also allow rotation; we study the placement of a triangle in a convex polygon. This can be regarded as a warm-up exercise that sheds some light on the general and more complex algorithm. In Section 5 we state some necessary definitions and results from [19]. In Section 6 we describe a variant of the fixed size containment algorithm from [17], which we use as a decision procedure, to decide whether a copy of P having some fixed expansion ratio can be placed in Q. In Section 7 we give a parallel version of the fixed size containment algorithm. In Section 8 we show how to combine the algorithms of Sections 6 and 7 to produce an algorithm for the largest placement problem. We conclude with a discussion of our results and some open problems.

1.1. Parametric search technique

We describe below the idea behind Megiddo's technique [21]. The exposition is driven by our needs, so it is not the most generally possible. Suppose we have a decision problem $\mathscr{P}(I, \delta)$ that receives as input a collection I of n data items, and a real parameter δ. Furthermore, assume that the decision is a monotone function of δ for every fixed input data, that is, there exists a real number $\delta^\star = \delta^\star(I)$ so that

$$\mathscr{P}(I, \delta) = \begin{cases} T & \delta \leqslant \delta^\star(I), \\ F & \delta > \delta^\star(I). \end{cases}$$

We want to find the value $\delta^\star$ for a given input I.

Assume that we have both an efficient sequential algorithm A_s for solving $\mathscr{P}(I, \delta)$ at any given δ, and a parallel algorithm A_p, assumed to run in Valiant's comparison-based model of parallel computation [27]. We will denote the running time of A_s by T_s, the running time of A_p by T_p, and the number of processors it uses by P. Assume

moreover that the flow of execution of A_p depends only on comparisons, each of which is resolved by testing the sign of a low-degree polynomial in δ and the input items. Megiddo's technique then runs the algorithm A_p 'generically', without specifying the value of δ, with the intention of simulating its execution at the unknown $\delta^\star$.

At each step of A_p, where there are at most P comparisons to resolve, all the (real) roots of all the associated polynomials are computed. If we knew between which two of them (in the natural ordering of the roots) $\delta^\star$ lies, we could resolve all the comparisons, because the sign of any of the polynomials is constant between consecutive roots, and continue to the next parallel step. So we search $\delta^\star$ in the list of $O(P)$ roots, using binary search and utilizing A_s to decide if a given root p_i is below or above $\delta^\star$ (i.e., we solve $\mathscr{P}(I, p_i)$). This search requires $O(P + T_s \log P)$ time per parallel step, for a total of $O(PT_p + T_sT_p \log P)$ time.

Through this execution of the generic A_p, we obtain a sequence of progressively smaller intervals, each known to contain $\delta^\star$. In our applications, the value $\delta^\star$ must eventually become a left endpoint of one of the intervals, and so at the end of the execution of A_p, the left endpoint of the resulting interval is $\delta^\star$. A more detailed discussion is presented later.

2. Placement of polygons under translation

In this section we investigate problems of placement of the largest homothetic copies of polygons inside another polygon (i.e. allowing translations and scaling only). In this case the parametric searching technique does not always yield the best solutions, and we will describe in the next section slightly more efficient solutions to some of these problems. Nevertheless, we present the solutions based on parametric searching in order to illustrate the technique for relatively simple problems, before tackling the general extremal polygon placement problem.

Definitions. We denote the set of translations of P that place it inside Q by $\mathscr{C}(P, Q)$, and the set of translations of P that make it intersect Q by $\mathscr{O}(P, Q)$, the latter operation is also known as the *Minkowski sum* of Q and $-P$.

We will assume that the polygons P and Q are given as an array of their vertices in counterclockwise direction, $P = (p_1, \ldots, p_k)$ and $Q = (q_1, \ldots, q_n)$. We will consider Q as fixed and P as movable, and we will use the vertex p_1 as a reference point for P.

Proposition 2.1 ([6]). *If P and Q are convex, then $\mathscr{C}(P, Q)$ is a convex polygon with at most n edges.*

Proposition 2.2 ([15]). *If P and Q are convex, then $\mathscr{O}(P, Q)$ is a convex polygon with at most $n + k$ edges.*

2.1. Computation of $\mathscr{C}(P,Q)$

We assume that both P and Q are convex. The procedure given below for the computation of $\mathscr{C}(P, Q)$ is taken from Chazelle [6].

1. For each edge q_iq_{i+1} of Q, we find the vertex p_i of P that is nearest to the line containing q_iq_{i+1}, when P lies completely on the same side of the line q_iq_{i+1} as Q (there may be two such vertices, in which case we choose one of them arbitrarily).

   This may be done in time $O(n + k)$ by merging the normal diagrams of P and Q, i.e., merging the edges of P and Q to a single list sorted by slope, and finding for each edge of Q between which edges of P it lies in the merged list.
2. For every $i = 1, 2, \ldots, n$ we place P so that p_i lies on the edge q_iq_{i+1}, and P and Q lie on the same side of the line q_iq_{i+1}. Now we draw a line t_i parallel to q_iq_{i+1} and passing through the reference point p_1 of P.

   Let ht_i denote the half plane that lies below t_i if Q lies below the line q_iq_{i+1} and above t_i otherwise. The computation of all the ht_i takes $O(n)$ time, a constant time for each half plane.
3. As shown in [6], $\mathscr{C}(P, Q) = \bigcap_i ht_i$, so what remains to do is to compute the intersection of the n half planes. We note that the half planes are given sorted by their slope. We compute the intersection by solving the dual convex hull problem, and the sorting of half planes by slope gives us a convex hull problem of n points sorted by their x-coordinate.

   This problem can be solved in $O(n)$ time, using standard techniques, e.g., the Graham Scan.

We conclude that the computation of $\mathscr{C}(P, Q)$ can be done in $O(n + k)$ time.

In order to apply the parametric search technique of Megiddo, we need a parallel version of this algorithm. Step 1, the merging of the normal diagrams, could be performed in parallel in $O(\log\log(\min(n, k)))$ parallel time using $\sqrt{nk}$ processors, using Valiant's algorithm [27]. However, the normal diagrams of P and Q are independent of the expansion ratio, so no comparison that this merge generates depends on $\delta^\star$. We can thus implement this step sequentially, 'outside' the generic scheme of Megiddo. Step 2 involves no comparisons, so it too can be performed sequentially. The coefficients of the t_i's will be however functions of the expansion ratio of P. Step 3 is performed in parallel using the parallel algorithm for computing the convex hull of plane point set, by Aggarwal et al. [3], that works in $O(\log n)$ time and uses $O(n)$ processors.

We now combine the sequential and parallel algorithms to obtain an algorithm that computes the largest homothetic copy of P that can be placed in Q. Note that the problem at hand satisfies the requirements of Megiddo's technique, that is, when the expansion ratio δ is smaller than some (unknown) value $\delta^\star$, there is a placement of P inside Q, and when $\delta > \delta^\star$ there is no such placement. We run the generic parallel algorithm, without specifying δ. We resolve comparisons needed by the algorithm by

computing the set of real roots of the characteristic polynomials associated with the comparisons, and locating $\delta^\star$ in this (ordered) set by binary search. The decisions made during the binary search are based on the outcome of the fixed-size algorithm, applied to a copy of P with expansion ratio equal to the root δ being compared. Note that the decision step only tells us whether $\delta \geqslant \delta^\star$ or $\delta < \delta^\star$. In order not to get stuck, we interpret $\delta \geqslant \delta^\star$ as $\delta > \delta^\star$ and continue in this manner. When the entire algorithm terminates, it will have produced an interval I so that $\delta^\star$ is either its left endpoint or an interior point. However, the second case is impossible, because the output of the generic algorithm is the same for all $\delta \in \text{int}(I)$, but the output must change at $\delta^\star$, by definition. Hence $\delta^\star$ is the left endpoint of I.

The running time of the algorithm is $O(n + k)$, for the initial step 1 performed just once, plus the cost of the parametric search itself, which is $O(n \log^2 n)$. We thus obtain the following.

Theorem 1. *Given a convex polygon P with k vertices and a convex polygon Q with n vertices, we can compute a placement of the largest homothetic copy of P inside Q in $O(k + n\log^2 n)$ time.*

Remark. As noted by Chazelle [6], this will work even if P is not convex, because in this case we simply apply our algorithm to $\text{conv}(P)$ instead of P.

Remark. In Section 3 we show how to improve the running time to linear.

2.2. Computation of $\mathcal{O}(P, Q)$

As shown by Guibas et al. [15], the calculation of $\mathcal{O}(P, Q)$ in the case of two convex polygons P and Q amounts to the merging of the lists of their edges sorted by slope.

This takes time $O(n + k)$ using a serial algorithm, or $O(\log\log(\min(n, k)))$ parallel time using $\sqrt{nk}$ processors, using Valiant's algorithm [27].

2.3. Finding largest homothetic placements of two convex polygons inside a third

We now consider the following problem. Given two convex polygons P_1 and P_2 having k_1 and k_2 vertices respectively, and a third convex polygon Q having n vertices, find the largest expansion ratio α such that αP_1 and αP_2 can be translated into Q without overlapping each other.

For the fixed-size containment problem we use the procedure given by Avnaim and Boissonnat [4] and Guibas et al. [15]. The procedure computes the set U of all the valid translations T_r of P_1 relative to P_2, for which there exists a translation that positions both P_1 and P_2 in Q in their valid relative position without overlapping. This

is done by several consecutive applications of the primitive operations $\mathscr{C}$ and $\mathscr{O}$:

1. Compute $C_1 = \mathscr{C}(P_1, Q)$.
2. Compute $C_2 = \mathscr{C}(P_2, Q)$.
3. If C_1 or C_2 is empty then return $\emptyset$.
4. Compute $I = \mathscr{O}(P_2, P_1)$.
5. Compute $S = \mathscr{O}(C_2, C_1)$.
6. Compute $S^\star$, the polygon symmetric to S with respect to the origin.
7. Return $U = S^\star \setminus I$.

The correctness of this algorithm is proved in [4], and we repeat it here for the sake of completeness.

Lemma 2.1 ([4]). *The set of all the valid translations of P_1 relative to P_2 is given by $U = S^\star \setminus I$.*

Proof. Assume that u is a valid relative translation of P_1. We have to show that $u \in U$. If u is valid, then P_1 and P_2 do not collide, so certainly $u \notin I$. However as u is a valid relative translation, it is the difference between two valid absolute translations $u = u_1 - u_2$, where u_1 is a translation of P_1 and u_2 a translation of P_2. Since these are valid absolute translations, $u_1 \in C_1$ and $u_2 \in C_2$. In addition, if we translate P_2 by $-u$, we should be able to translate both polygons into Q using the translation u_1, and this means that C_2 translated by $-u$ should intersect C_1, or $-u \in S = \mathscr{O}(C_2, C_1)$. Thus $u \in S^\star$ but $u \notin I$. This argument, applied in reverse, shows the other direction as well and the proof is complete. □

From the proposition above and the descriptions of algorithms for the computations of $\mathscr{C}$ and $\mathscr{O}$, it follows that the running time of this algorithm is $O(n + k_1 + k_2)$.

The parallel version of the algorithms for computing $\mathscr{C}$ and $\mathscr{O}$ can be used for performing Steps 1–5 of the algorithm above. Step 6 does not involve comparisons, so we need not perform it in parallel. Step 7 is more difficult to handle, but we exploit the fact that we are only interested in the existence of a translation in U, not in its full structure. So instead of computing U, we will only decide in Step 7 whether $U = S^\star \setminus I$ is empty or not. As both $S^\star$ and I are convex polygons, the difference is not empty if and only if the convex hull of $S^\star \cup I$ is not simply I. This is so because $S^\star \setminus I$ is empty if and only if $S^\star$ is contained in I, and this is true if and only if the convex hull of their union is I. So computing the convex hull of $S^\star \cup I$ is sufficient to decide on the non-emptiness of U, and this computation can be performed in $O(\log(n + k_1 + k_2))$ parallel time using $O(n + k_1 + k_2)$ processors.

Applying the parametric search paradigm, we obtain the following.

Theorem 2. *Given two convex polygons P_1 and P_2 with k_1 and k_2 vertices respectively, and a convex polygon Q with n vertices we can compute pairwise-disjoint placements of*

the largest homothetic copies of P_1 and P_2 inside Q (with the same expansion ratio), in $O((n + k_1 + k_2)\log^2(n + k_1 + k_2))$ time.

The assumption that the expansion of P_1 and P_2 is the same is not necessary; we only have to assume that the expansion ratios of the two polygons are expressed by two monotone increasing functions of the same parameter, $f_1(\alpha)P_1$, $f_2(\alpha)P_2$.

3. Improved techniques for polygon placement under translation

3.1. Placing polygons using linear programming

In cases where we have to place a translated copy of one polygon in another convex polygon a more efficient approach can be taken. In these cases we can reduce polygon placement problems to linear programming problems. We are indebted to Nimrod Megiddo and independently to Alon Efrat for pointing this out.

We are given two polygons, a convex polygon $Q = (q_1, \ldots, q_n)$ and a simple polygon $P = (p_1, \ldots, p_k)$. Without loss of generality assume that P is also convex (otherwise replace P with conv (P)). We have to place a copy of P inside Q. The transformations allowed (translation, translation and scaling etc.) will be specified later.

Assume that we seek a placement of the largest homothetic copy of P in Q. The copy of P is thus $\alpha P + b$ where $\alpha > 0$ is a real expansion ratio and b is a translation vector. As in Subsection 2.1 we only need to ensure that P is on the 'right' side of each edge of Q, and this holds if the vertex of P nearest to the line containing that edge is on the 'right' side of the edge (the edge-vertex pairing can be computed in time $O(n + k)$). That is, we will have n inequalities that ensure that the copy of P is inside Q,

$$a_i^{\mathrm{T}}(\alpha p_i + b) \leqslant 1$$

for $i = 1, \ldots, n$, with the linear objective function max α. The a_i's and p_i's are constants. The variables are the vector b and the expansion-ratio α.

This linear programming problem can be solved in $O(n)$ time using Megiddo's linear programming algorithm [22, 23] and the overall running time is thus $O(n + k)$.

The technique can also be applied to 3-dimensional convex polyhedra. As before, we find the nearest vertex of P for each plane containing a face of Q, and solve the linear programming problem

$$a_i^{\mathrm{T}}(\alpha p_i + b) \leqslant 1$$

for $i = 1, \ldots, n$, with the linear objective function max α.

Finding the nearest vertices of P to the planes containing the faces of Q can be done by computing the normal diagram of P (which is actually a planar map on a sphere), preprocessing it for fast point location, and locating the normal direction of each face of Q in this map. Computing the normal diagram can be done in linear time, given a reasonable representation of the polyhedra (e.g. a quad-edge structure), preprocessing

can be done in $O(k)$ time using the technique of [13], and the point location queries can be all done in $O(n \log k)$ time. Thus the overall running time of the algorithm is $O(k + n \log k)$.

It is obvious that the technique can be applied to fixed-size queries as well. In this case only the translation vector b is unknown, and we just seek a feasible solution for b.

We thus obtain the following result.

Theorem 3. *Given a convex polygon P with k vertices and a convex polygon Q with n vertices, we can compute a placement of the largest homothetic copy of P inside Q in $O(k + n)$ time. In three dimensions, we can do the same for convex polytopes in time $O(k + n \log k)$.*

3.2. Placing two convex polygons

In this subsection we present an improved solution for the problem of finding the largest homothetic placement of two convex polygons P_1, P_2 inside a third convex polygon Q, as studied in Subsection 2.3. We are indebted to an anonymous referee for suggesting this solution.

For a given expansion factor α, a placement of αP_1 and αP_2 inside Q is possible if and only if $S_\alpha^\star \setminus I_\alpha$ is nonempty, where

$$S_\alpha^\star = -\mathcal{O}(\mathcal{C}(\alpha P_2, Q), \mathcal{C}(\alpha P_1, Q))$$

and

$$I_\alpha = \mathcal{C}(\alpha P_2, \alpha P_1) = \alpha \mathcal{C}(P_2, P_1).$$

The idea of the new approach is to construct $S_\alpha^\star$ and I_α for all α simultaneously, by adding α as a third coordinate.

The set $I = \{(x, y, \alpha);\ \alpha \geqslant 0,\ (x, y) \in I_\alpha\}$ is clearly a convex polytope, which can be constructed in linear time. Consider the set $C_1 = \{(x, y, \alpha);\ \alpha \geqslant 0,\ (x, y) \in \mathcal{C}(\alpha P_1, Q)\}$. As shown above, $\mathcal{C}(\alpha P_1, Q)$, for any fixed α, is defined as an intersection of n half planes, and as α varies these half planes span a half space bounded by some plane, as is easily seen. Hence $\mathcal{C}(\alpha P_1, Q)$ is also a convex polytope in 3-space, which can be computed in $O(n \log n)$ time. Finally we construct the 'planewise Minkowski sum'

$$S = \{(x, y, \alpha);\ \alpha \geqslant 0,\ (x, y) \in \mathcal{O}(\mathcal{C}(\alpha P_2, Q), \mathcal{C}(\alpha P_1, Q))\}.$$

We note that S is also a convex polytope in 3-space. To see this, observe that, if we vary α in some interval I, so that no vertex of either $\mathcal{C}(\alpha P_1, Q)$ or $\mathcal{C}(\alpha P_2, Q)$ has α-coordinate in I, then the combinatorial structure of all the planar cross-sections $\mathcal{O}(\mathcal{C}(\alpha P_2, Q), \mathcal{C}(\alpha P_1, Q))$ remains the same, the orientation of each edge of this convex polygon does not change as α varies, and each vertex of this polygon moves along a straight line at constant velocity as α varies. This is easily seen to imply that S is also a convex polytope of linear complexity, and that it can be constructed in $O(n \log n)$

time, by sweeping a plane in the α-direction, and by updating S every time the plane passes through a vertex of either $\mathscr{C}(\alpha P_1, Q)$ or $\mathscr{C}(\alpha P_2, Q)$. Now we can compute $(-S) \cap I$ (in linear time by the algorithm of Chazelle [8]), and find out the largest α for which the horizontal cross-section of $(-S) \cap I$ is equal to that of $(-S)$; this gives the largest expansion factor of P_1 and P_2 for which they can still be placed inside Q, as is easily verified.

The running time of this algorithm is $O(n \log n + k_1 + k_2)$, which is better than the algorithm given in Subsection 2.3. Nevertheless, the former algorithm is easier to design and, as remarked above, it also applies to a somewhat more general situation.

4. Placing a triangle in a convex n-gon under translation and rotation

Before we tackle the general problem of extremal containment of a convex polygon in a general polygonal environment, we consider a restricted version in which we compute the largest similar copy of a triangle $T = ABC$ in a convex polygon $Q = (q_1, \ldots, q_n)$.

This (fixed size) containment problem was studied by Chazelle [6]. He observed that there is a free placement of T in Q if and only if there is a placement of T in Q in which a vertex of T and a vertex of Q coincide. Thus in order to test if there exists a free placement of T in Q, we go over all the $3n$ pairs of a vertex of T and a vertex of Q and for each pair test if there is a free placement such that the relevant vertices coincide.

When the vertices of such a pair, say A and q_1, coincide, we use the angle θ of rotation around the fixed vertex A of T to describe the placement of T. The placement is free iff both edges AB and AC lie in all the half-planes whose intersection is Q. As B and C can intersect the line defining a half-plane only twice when T rotates around A, we can generate a pair of intervals of placements (= angles) that are free relative to that half-plane. The intersection of all n intervals is the set of free placements. This intersection can be computed in time $O(n \log n)$ per pair, or $O(n^2 \log n)$ overall.

The parallel version works by sorting all the endpoints of all free intervals for each vertex-vertex contact. This takes $O(\log n)$ time and uses $O(n)$ processors per pair [10], or $O(\log n)$ time and $O(n^2)$ processors overall. Using Cole's improvement to Megiddo's technique [11, 21] we can save a logarithmic factor in the running time and obtain the following.

Theorem 4. *Given a triangle T and a convex polygon Q with n vertices, we can compute a placement of the largest possible similar copy of T inside Q in time $O(n^2 \log^2 n)$.*

The discussion above is similar in nature to the solution of the general case given below. The increased complexity caused by allowing rotations prevents us from computing the set of all possible free placements as we did when translation alone was allowed. Instead we restrict our attention to a distinguished subset of 'critical' free placements that necessarily exist if any free placement exists. There is also an analogy

between computing the intersection of relatively free intervals to find a free placement, and the use of lower envelopes below. The details of the general case, however, are much more complex.

5. The general case—finding free critical orientations

We now begin the description of our solution to the general case. In this section we give a short exposition of the definitions and results in [19]; this is needed in order to present the algorithms in subsequent sections. The material is taken almost verbatim from [19].

Let P be a convex polygonal object having k vertices, free to translate and rotate (but not to change its size) in a closed two-dimensional space Q bounded by a collection of polygonal obstacles ('walls') having altogether n corners.

A *free critical* placement of P is one at which it makes simultaneously three distinct contacts with the walls, and is fully contained in Q, so that it cannot penetrate any obstacle.

A (potential) *contact pair* O is a pair (W, S) such that either W is a (closed) wall edge and S is a corner of P or W is a wall corner and S is a (closed) side of P. The contact pair is said to be of type I in the first case, and of type II in the second case.

An actual *obstacle contact* is said to *involve* the contact pair $O = (W, S)$ if this contact is of a point on S against a point on W, and, furthermore, if this contact is *locally free*, i.e., the inner angle of P at S lies entirely on the exterior side of W if S is a corner of P, and the entire angle within the wall region Q^c at W lies exterior to P if W is a wall corner.

The *tangent line* T of a contact pair $O = (W, S)$ is either the line passing through W if W is a wall edge or the line passing through W and parallel to S if S is a side of P (in the second case T depends of course on the orientation of P).

Let O_1, O_2 be two contact pairs. We say that O_2 *bounds* O_1 at the orientation θ if the following conditions hold (see Fig. 1).

(1) There exists a (not necessarily free) placement $Z = (X, \theta)$ of P at which it makes two simultaneous obstacle contacts involving O_1, O_2.

(2) As we move P from Z without changing the orientation θ, along the tangent T_1, in the direction of the intersection z of the two tangents T_1 and T_2, the subset $P^\star = \mathrm{conv}(S_1 \cup S_2)$ of P intersects W_2 until S_1 touches W_1.

It is shown in [19] that for any double obstacle contact, one of the contact pairs always bounds the other. Let O_1 be any contact pair and consider all contact pairs that bound O_1 (at any orientation θ). For each such pair O_2 we define the *bounding function* $F_{O_1,O_2}(\theta)$ over the domain $\Pi = \Pi_{O_1,O_2}$ of orientations θ of P in which O_2 bounds O_1. For each $\theta \in \Pi$, we define $F_{O_1,O_2}(\theta)$ to be the distance from the endpoint of the contact wall farthest from z (the intersection of the tangents) to the contact

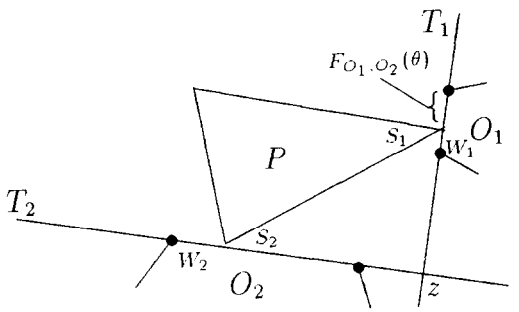

Fig. 1. A bounding function.

point of O_1, at the placement $Z = (X, \theta)$ in which P simultaneously makes two obstacle contacts involving O_1, O_2 (see Fig. 1). Note that Π need not be connected, but it consists of at most five subintervals (this is proved in [19, Lemma 2.2]).

The dependence of the bounding function on a specific endpoint of the contact wall suggests that we group the bounding functions F_{O_1,O_2} of O_1 into two classes, A_L and A_R, so that in each class the functions are related to the same endpoint of the contact wall of O_1.

With each class A_E, $E \in \{L, R\}$, of each contact pair O_1, we associate a function

$$\Psi_{E;O_1}(\theta) = \min \{F_{O_1,O}(\theta) : F_{O_1,O} \in A_E\}.$$

This is the *lower envelope* of the functions in A_E. An intersection of two bounding functions of the same class, F_{O_1,O_2} and F_{O_1,O_3}, that lies on the lower envelope of that class, is called a *breakpoint* of the lower envelope.

Critical free orientations (i.e. orientations of critical free placements) can arise in three situations. The *first kind* of orientations is of critical placements at which two contact pairs simultaneously bound a third one, and both belong to the same class. Each such placement is represented as a breakpoint on some lower envelope. The *second kind* of orientations arise at critical placements where two contact pairs bound a third one but belong to different classes. The *third kind* of orientations arise when no two contact pairs bound a third, but rather at critical placements involving three contact pairs O_1, O_2 and O_3 so that O_1 bounds O_2, O_2 bounds O_3, and O_3 bounds O_1. See Fig. 2 for an illustration.

These are necessary conditions for a critical free placement of P, that is, one of the three situations must occur at a critical free placement. However, they are not sufficient, and while our algorithm will find every orientation of any of the three kinds, it must also be able to discard critical placements that are not free.

It is proved in [19] that the number of breakpoints along one lower envelope is $O(\lambda_6(kn))$. This implies that the total number of critical orientations, of all three kinds, is $O(kn\lambda_6(kn))$.

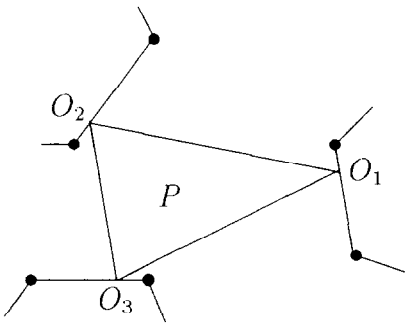

Fig. 2. A critical contact of the third kind.

Remark. The analysis of [19], when turned into an algorithm, can produce a list of all these critical placements in time $O(kn\lambda_6(kn)\log(kn))$. However, detecting which of these placements is indeed free is not straightforward. In the context of motion planning, as studied in [17], it is possible to sift out the critical orientations and obtain a subset of *free* critical placements that include all placements reachable from a given initial placement, and perhaps some other non-reachable but free placements. This can be done within the same time bound, $O(kn\lambda_6(kn)\log(kn))$, but cannot guarantee that *all* free placements are found, and is therefore unsuitable for our purpose. This revised version of the algorithm of [17] is presented in a recent paper of Kedem et al. [18]; see also a discussion in the algorithms that we give below. In our solution, we do detect all free critical placements, at an extra cost of $O(k\log n)$ per placement. Performing this faster still appears to be an open problem.

6. A Sequential algorithm

In this section we present a sequential algorithm for the fixed size containment problem, that is to determine whether it is possible to place a similar copy of the convex polygonal object P, at some fixed expansion ratio to the original P, in the polygonal environment Q. To solve this decision problem we solve a related problem—finding all the *critical free orientations* of P. If the set of critical free orientations is empty, the solution to our decision problem is 'no', otherwise the answer is 'yes'.

6.1. Generating all the critical placements

Below we give the algorithm that generates all the critical placements. Each one of them needs to be tested to decide whether it is free, using the algorithm of the next subsection.

The algorithm closely follows the first stages of the algorithm in [17]. However, the data structures used are simpler, to ease the task of parallelizing the algorithm later. We do not consider critical contacts in which P has only one degree of freedom

(a corner of P against a corner of Q, an edge of P against an edge of Q), because they can be handled exactly like the triangle in Section 4, that also has one degree of freedom in every critical placement.

Step 1: *Find all bounding functions.*

For every two contact pairs O_i, O_j, find the range of orientations Π_{O_i,O_j} in which O_j bounds O_i toward a specific endpoint E of O_i. Split the resulting bounding functions F_{O_i,O_j} into (at most five) 'subfunctions', each defined over a connected interval, and add them to the appropriate collection $A_{\mathrm{L}}(O_i)$ or $A_{\mathrm{R}}(O_i)$.

Step 2: *Calculate lower envelopes.*

We describe the calculation of the lower envelope of $A_L(O)$ which is denoted by $\Psi_{\mathrm{L};O}$; $\Psi_{\mathrm{R};O}$ is calculated similarly.

1. Fix a contact pair O and partition $A_{\mathrm{L}}(O)$ into two disjoint subsets A'_{L} and A''_{L} of roughly equal size.
2. Compute recursively the two lower envelopes

$$\Psi'(\theta) = \min\{F_{O,O_i}(\theta) : F_{O,O_i} \in A'_{\mathrm{L}}\},$$

$$\Psi''(\theta) = \min\{F_{O,O_i}(\theta) : F_{O,O_i} \in A''_{\mathrm{L}}\}.$$

   Each of the recursive calculations produces a sequence of angular intervals, delimited by breakpoints, in each of which the corresponding partial lower envelope is attained by a single bounding function.
3. Merge these two sequences of intervals to obtain a refined sequence Γ of angular intervals. In the merging process mark every breakpoint in Γ as red if it was originally a breakpoint of Ψ' or as black if it was originally a breakpoint of Ψ''. In addition, maintain a pointer from each red node in Γ to the black interval it lies in (an interval of Ψ'') and from each black node to the red interval it lies in. For each interval $I \in \Gamma$ there exist two unique contact pairs O', O'' with $F_{O,O'} \in A'_{\mathrm{L}}$, $F_{O,O''} \in A''_{\mathrm{L}}$ such that $\Psi'(\theta) = F_{O,O'}(\theta)$, $\Psi''(\theta) = F_{O,O''}(\theta)$ for each $\theta \in I$. By the analysis of [19] the two functions $F_{O,O'}$, $F_{O,O''}$ intersect in at most four points (some of which may not belong to I), which can be calculated, as the roots of some quartic polynomial, in constant time. Each of these intersections which lies in I is clearly a breakpoint of $\Psi = \Psi_{\mathrm{L};O}$. Add these points to Γ and mark them as white nodes. Every breakpoint of Ψ is either of this kind (a white node) or is a breakpoint of Ψ' or of Ψ'', i.e. one of the red or black nodes. Now we need to eliminate from Γ the red and black nodes which do not lie on the lower envelope Ψ. For each red (black) node, we follow the pointer to the black (red) interval it lies in, and check which is higher—the red breakpoint in Ψ' or the bounding function on Ψ''. If the former is higher we prune it from the list Γ, otherwise the breakpoint remains in Γ. Thus at the end of the process Γ represents the breakpoints in Ψ.

Note that maintaining the red/black pointers can be done in time proportional to the length of the list (in one pass over the list), and the same time bound applies to the pruning of the redundant nodes.

The merging step can be done in time proportional to the length of Γ, which, by [19] and the comments in Section 2, is $O(\lambda_6(kn))$. Hence the calculation of the lower envelope $\Psi_{L;O}$ takes $O(\lambda_6(kn)\log kn)$ time, so all these envelopes can be computed in overall time $O(kn\lambda_6(kn)\log kn)$.

The collection of breakpoints is a superset of all the critical orientations of the first kind; every one of them will later be tested to decide whether it is free, in the manner described in the next subsection.

Step 3: *Calculate critical orientations of the second kind.*

These are orientations at which P makes simultaneously, at some free placement, obstacle contacts involving three distinct contact pairs O_1, O_2, O_3 such that two of them, say O_2, O_3 bound O_1 but with $F_{O_1,O_2} \in A_L(O_1)$ while $F_{O_1,O_3} \in A_R(O_1)$. In this case we first reflect and translate one of the envelopes, so that they both measure the distance from the same endpoint of O_1. Then we merge the lists of breakpoints in $\Psi_{L;O_1}$ and in $\Psi_{R;O_1}$ and compute the intersections of the bounding functions from the two lower envelopes over each resulting interval in the same way as in the previous step. These orientations are added to the list of critical orientations.

Clearly, this step runs in $O(kn\lambda_6(kn))$ time. Again, we will later discard non-free critical orientations found in this step.

Step 4: *Calculate critical orientations of the third kind.*

Finally, we calculate the third and most complex kind of critical orientations. At each such orientation θ, P can make simultaneously a free triple contact involving three distinct contact pairs O_1, O_2, O_3, such that $F_{O_1,O_2} \in A_{E_1}(O_1)$, $F_{O_2,O_3} \in A_{E_2}(O_2)$, $F_{O_3,O_1} \in A_{E_3}(O_3)$, where $E_i \in \{L, R\}$ for $i = 1, 2, 3$, and such that all three functions lie at θ on the corresponding lower envelopes.

To find these orientations we first merge all breakpoint lists from all the lower envelopes calculated in Step 2, to obtain a single sorted list Φ consisting of $O(kn\lambda_6(kn))$ refined non-critical intervals. Each interval $I \in \Phi$ has the property that each lower envelope is attained over it by a single bounding function.

Next we find all the critical orientations of the third kind (not necessarily free). For each possible triple contact that the algorithm considers, we find its (at most four) critical orientations, and then test which of these orientations is indeed free.

We start by considering the first interval in Φ, denoted I_0. For each contact pair O_1 and each side $E_1 \in \{L, R\}$, find the unique contact pair O_2 such that $\Psi_{E_1;O_1} = F_{O_1,O_2}$ over I_0. For each $E_2 \in \{L, R\}$, find the unique contact pair O_3 such that $\Psi_{E_2;O_2} = F_{O_2,O_3}$ over I_0. For each $E_3 \in \{L, R\}$ for which $\Psi_{E_3;O_3} = F_{O_3,O_1}$ over I_0 conclude that $(O_1, O_2, O_3, E_1, E_2, E_3)$ is a critical contact, perhaps not free. Compute its critical

orientations. Those that lie in I_0 are tested to decide whether they are free, and if so they are reported as such (the algorithm can thus be halted right now with an affirmative answer to the decision problem). The other orientations do not lie in I_0, so we find the interval each of them lies in, by binary search over the sorted list Φ, and test whether the corresponding critical placement is free. This takes $O(kn \log(kn))$ time, excluding the tests for being free, $O(\log(kn))$ time for each contact pair.

Each interval $I \neq I_0$ in Φ can induce new critical triplets but fortunately only a constant number of them. The interval I was formed because its left endpoint represents a break in one of the lower envelopes, say $\Psi_{L;O_1}$. So we need to repeat the process we did at I_0, but this time starting with only one particular contact (O_1) and one lower envelope ($\Psi_{L;O_1}$). Thus every interval I induces only $O(1)$ new candidates for the critical orientations that we seek. Finding the bounding functions on the lower relevant envelopes is now accomplished by a binary search over the list of breakpoints on each envelope. This takes $O(kn\lambda_6(kn) \log(kn))$ time, as each of the $O(kn\lambda_6(kn))$ intervals requires $O(\log(kn))$ time for the binary searches.

6.2. *Deciding whether a critical orientation represents a free placement*

As mentioned above, the set of critical orientations computed so far may contain orientations that correspond to critical placements that are not free, so we need to test each critical placement whether it is indeed free.

To perform this test we use the following simple method. In a preliminary step, we prepare data structures that will enable us to perform this test, at any query placement of P, in time $O(k \log n)$. These data structures depend only on Q. As we are required to perform at most $O(kn\lambda_6(kn))$ such tests, the total time they require is $O(k^2 n\lambda_6(kn) \log n)$. As the environment Q is static during the execution of the (largest placement) algorithm, we need to build the data structures only once, 'outside' the generic execution of the parallel version of the algorithm.

If a critical placement is not free, then either a vertex of Q lies inside P, or an edge of P intersects an edge of Q. To test whether the first situation occurs we insert the n vertices of Q into a data structure that supports fast counting of points inside a query polygon. We use the technique of [24, 12], which uses $O(n^2)$ storage, $O(n^2)$ preprocessing, and can answer a query in time $O(k \log n)$ for a query polygon with k sides. Given a placement of P, we query the number of points inside it and declare the placement non-free if any such point is found. To test whether the second situation arises, we preprocess Q for segment intersection queries, that is, given a query segment, determine quickly whether it intersects an edge of Q. For this we use the technique of [7]. Again, this can be implemented with $O(n^2)$ storage, $O(n^2)$ preprocessing, and can answer a segment intersection query in $O(\log n)$ time. For each critical placement of P, we query this structure with each edge of P, and declare the placement as non-free if any such intersection is found. If none of these bad situations are detected, the placement is free.

We have thus shown the following.

Theorem 5. *Given a convex polygon P with k sides, and a polygonal environment Q with n edges, we can compute all free critical placements of P inside Q in time* $O(k^2 n \lambda_6(kn) \log(kn))$.

7. A parallel algorithm

We now present a parallel version of the algorithm, or rather comment on how to perform each step in parallel. Recall that we do not need a strong parallel computation model. All we seek is a scheme in which many independent comparisons are performed at each parallel step. Thus we ignore synchronization and other bookkeeping problems, use Valiant's weak model of parallel computation [27], and perform tasks in a sequential manner when they do not involve comparisons but only manipulation of pointers.

Step 1 can clearly be carried out by $O(k^2 n^2)$ processors in $O(1)$ parallel time, with each processor calculating one bounding function.

Step 2 is performed using a divide and conquer strategy. The divide phase and the recursive calls can be done in parallel. The merge phase can be done using Valiant's merging algorithm [27] which runs in $O(\log\log(kn))$ parallel time using $O(\lambda_6(kn))$ processors per envelope. Once the merge is done, maintenance of the red/black pointers can be done serially because this is a mere manipulation of pointers and involves no comparisons. The testing of red (black) breakpoints against their containing black (red) non-critical intervals can be done in $O(1)$ parallel time using $O(\lambda_6(kn))$ processors per envelope. The subsequent pruning of breakpoints can be done serially, because it involves no comparisons. The total time required to compute all the lower envelopes is thus $O(\log(kn)\log\log(kn))$ using $O(kn\lambda_6(kn))$ processors.

Step 3 also uses a merge, but only once for each pair, and the calculation of envelope intersections over all contact pairs can clearly be done in parallel. The total parallel time for this step is therefore $O(\log\log(kn))$ using $O(kn\lambda_6(kn))$ processors.

Step 4 first requires the merging of all $O(kn)$ lower envelopes into one sorted list of breakpoints Φ. This can be done recursively. We divide the lower envelopes into two collections of roughly equal size, compute the two merged lists of breakpoints Φ', Φ'' and merge them. The merging steps takes $O(\log\log(kn))$ parallel time using $O(kn\lambda_6(kn))$ processors, so the whole process takes $O(\log(kn)\log\log(kn))$ time using $O(kn\lambda_6(kn))$ processors.

The handling of the first interval $I_0 \in \Phi$ can be done in parallel using $O(kn)$ processors and $O(\log(kn))$ time. All the other intervals are each assigned a single processor and the time it takes to find the new critical triple contacts is $O(\log(kn))$. The test to decide whether a critical orientation is free is performed in the same manner as in the sequential case. We use k processors to perform the $O(k)$ queries in

$O(\log n)$ time. The data structures depend only on Q and so can be preprocessed just once, outside the generic parallel scheme.

We conclude that all critical free orientations can be calculated in parallel, under Valiant's comparison model, in time $O(\log(kn)\log\log(kn))$ using $O(k^2n\lambda_6(kn))$ processors.

8. The overall algorithm

We now apply Megiddo's technique to our problem, using the algorithms of Sections 6 and 7. We run the parallel algorithm generically, without specifying the expansion ratio δ. We resolve comparisons made by the parallel algorithm by using our sequential algorithm, in the manner explained in Subsection 1.1.

The only fine point is to verify that comparisons involve only evaluations of signs of low degree polynomials in the unspecified δ. Indeed, a free placement of P in Q has to satisfy a set of algebraic constraints (see [25]). In our case these constraints are mainly algebraic inequalities that describe the disjointness of P and Q. Computing a critical triple contact amounts to setting three inequalities as tight constraints (i.e. equalities), and solving these three equations in three unknowns (the (x, y, θ) coordinates of P), discarding solutions which are not locally free. Computing a breakpoint thus reduces to computing the critical triple contact placement associated with the three contact pairs that define the breakpoint. Evaluating a bounding function at a given orientation amounts to setting the two constraints involved in the corresponding two contact pairs to be tight, and adding a third constraint that the slope of the line passing between two fixed points in P will be at the given orientation. This will give us the desired placement of P, and we can calculate the value of the bounding function which is simply an affine transformation of the placement. Using the standard transformation $t = \tan(\theta/2)$, all contact constraints, and thus all functions of δ computed by the algorithm, become algebraic, and no trigonometric functions need be used.

Since the only place where dependence on δ can arise is in the coefficients of the constraints, and since the functions of δ are polynomials of the first or second degree, we are assured that all the equations in δ we have to solve during the algorithm are algebraic equations of bounded degree. We assume that this kind of equations can be handled in constant time.

The running time of the algorithm can easily be deduced by 'plugging in' the running time of the sequential and parallel algorithms into the analysis of Megiddo's technique. The fact that $\delta^\star$ is the left endpoint of the final interval is justified as in Subsection 2.1. This establishes our main result:

Theorem 6. *Given a convex polygon P with k sides, and a polygonal environment Q with n edges, we can compute a placement of the largest possible similar copy of P inside Q in time* $O(k^2n\lambda_6(kn)\log^3(kn)\log\log(kn))$.

9. Conclusions

In this paper we have applied Megiddo's parametric search technique to a variety of extremal polygon placement problems. In addition, we presented a decision algorithm for the general fixed-size polygon containment problem, which improves upon results obtained in previous papers that studied related problems.

Megiddo's technique can be applied to many other extremal containment problems. For example, Megiddo's technique is used in [2] to solve the problems of finding the largest stick (line segment) that can be placed in a simple polygon, and can also used to find a placement for a stick that minimizes the maximum distance from the stick to a given point-set in the plane.

Our work raises a few open problems. One is to improve our main algorithm by about an order of k, bringing its complexity close to that of the motion-planning algorithm of [17, 18]. Another question is whether our technique can be turned into a motion-planning algorithm that finds a 'highest-clearance' path among obstacles, as in [9].

References

[1] P. Agarwal, M. Sharir and P. Shor, Sharp upper and lower bounds on the length of general Davenport-Schinzel sequences, J. Combin. Theory Ser. A 52 (1989) 228–274.
[2] P.K. Agarwal, M. Sharir and S. Toledo, Applications of parametric searching in geometric optimization, Proc. 3rd ACM-SIAM Symp. on Discrete Algorithms (1992) 72–82.
[3] A. Aggarwal, B. Chazelle, L. Guibas, C. Ó'Dúnlaing and C. Yap, Parallel computational geometry, Algorithmica 3 (1988) 293–327.
[4] F. Avnaim and J.D. Boissonnat, The polygon containment problem: 1. Simultaneous containment under translation. Technical report 689, INRIA Sophia Antipolis, June 1987.
[5] F. Avnaim and J.D. Boissonnat, Polygon placement under translation and rotation, 5th Annual Symp. on Theoretical Aspects of Computer Science, Lectures Notes in Comp. Science 294 (Springer, New York, 1988) 322–333.
[6] B. Chazelle, The polygon containment problem, in: F.P. Preparata, ed., Advances in Computing Research, Vol. I: Computational Geometry (JAI Press, Greenwich, CT, 1983) 1–33.
[7] B. Chazelle, Reporting and counting segment intersections, J. Comp. Sys. Sci. 32 (1986) 156–182.
[8] B. Chazelle, An optimal algorithm for intersecting three-dimensional convex polyhedra, SIAM J. Comput. 21 (1992) 691–696.
[9] L.P. Chew and K. Kedem, A convex polygon among polygonal obstacles: placement and high-clearance motion, Comput. Geom. Theory Appl. 3 (1993) 59–89.
[10] R. Cole, Parallel merge sort, 27th IEEE Symp. on Foundations of Computer Science (1986) 511–516.
[11] R. Cole, Slowing down sorting networks to obtain faster sorting algorithms, J. Assoc. Comput. Mach. 34 (1987) 200–208.
[12] H. Edelsbrunner, Algorithms in Combinatorial Geometry (Springer, Berlin, 1987).
[13] H. Edelsbrunner, L.J. Guibas and J. Stolfi, Optimal point location in a monotone subdivision, SIAM J. Comput. 15 (1986) 317–340.
[14] S. Fortune, Fast algorithms for polygon containment, Proc. 12th International Colloquium on Automata, Languages and Programming, Lecture Notes in Comp. Science 194 (Springer, New York, 1985) 189–198.
[15] L. Guibas, L. Ramshaw and J. Stolfi, A kinetic framework for computational geometry, 24th IEEE Symp. on Foundations of Computer Science (1983) 100–111.

[16] S. Hart and M. Sharir, Nonlinearity of Davenport-Schinzel sequences and of generalized path compression schemes, Combinatorica, 6 (1986) 151–177.
[17] K. Kedem and M. Sharir, An efficient motion-planning algorithm for a convex polygonal object in two-dimensional polygonal space, Discrete Comput. Geom. 5 (1990) 43–75.
[18] K. Kedem, M. Sharir and S. Toledo, On critical orientations in the Kedem-Sharir motion planning algorithm for a convex polygon in the plane, Proc. 5th Canadian Conference on Computational Geometry (1993) 204–209.
[19] D. Leven and M. Sharir, On the number of critical free contacts of a convex polygonal object in two-dimensional polygonal space, Discrete Comput. Geom. 2 (1987) 255–270.
[20] D. Leven and M. Sharir, Planning a purely translational motion for a convex object in two-dimensional space using generalized Voronoi diagrams, Discrete Comput. Geom. 2 (1987) 9–31.
[21] N. Megiddo, Applying parallel computation in the design of serial algorithms, J. ACM 30 (1983) 852–856.
[22] N. Megiddo, Linear-time algorithms for linear programming in R^3 and related problems, SIAM J. Comput. 12 (1983) 759–776.
[23] N. Megiddo, Linear programming in linear time when the dimension is fixed, J. Assoc. Comput. Mach. 31 (1984) 114–127.
[24] M.S. Paterson and F.F. Yao, Point retrieval for polygons, J. Algorithms 7 (1986) 441–447.
[25] J.T. Schwartz and M. Sharir, On the Piano Movers Problem: II. General techniques for computing topological properties of real algebraic manifolds, Adv. in Appl. Math. 4 (1983) 298–351.
[26] S. Toledo, Extremal polygon containment problems, Proc. 7th ACM Symp. on Computational Geometry (1991) 176–185.
[27] L. Valiant, Parallelism in comparison problems, SIAM J. Comput. 4 (1975) 345–348.